The Legacy of Wisdom

Due: Sep 9, 2024 11:59 PM CST

Introduction:

Congratulations! You've just welcomed a new baby into your world. Determined to equip your child with life's most treasured wisdom and goodness, you decide to pen down a Life's Instruction Manual. This manual contains chapters filled with guidelines, constants, and truths of life. Let's start writing it using C++!

Instructions:

- Write a complete C++ program for this assignment. The program should be full-fledged and compiles and runs without errors.
- For file submission, it should just be a **SINGLE** \`.cpp\` file.
- Every file must include a proper comment header (name, date, file purpose, etc.)!

Chapter 1: Constants in Our Lives

In life, as in programming, there are certain universal truths that remain unchanging. These are the constants that guide our decisions and shape our understanding. In your program, declare *global constants* for these unchanging truths:

1. PI = 3.14159265;
2. EARTH_GRAVITY = 9.80665; // meters per seconds-squared
3. WISDOM = "Love, Kindness, and Integrity. And always eat your vegetables!";

Chapter 2: Piecing Together The Stories

You recall how your grandparents used to share snippets of life's tales. Let's pen down one of those tales.

1. Prompt the user: "How many stories did your grandparents share with you?"
2. Take input into an integer variable. Assume that this value ranges from 1 to 20.
3. Calculate the total duration of the stories in hours. Assume, for simplicity, each story is 5 hours and 20 minutes long.
4. In addition, calculate how many minutes are left over using the modulus (%) operator.
5. Example output:

```
    How many stories did your grandparents share with you? 8
The total storytelling time is 42 hours and 40 minutes.
```

Chapter 3: Drawing Life's Circles

The circle of life revolves around the constant of π (Pi).

1. Prompt the user: "Enter the diameter of the life's circle you want to draw."
2. Input this into a variable of floating point type.
3. Calculate the **circumference** and **area** using the constant PI.
4. Print out the result.

Chapter 4: Our Words and Actions

Every word we utter and every action we take, leaves an imprint.

1. Ask the user to enter a character and store it into a variable.
2. Display the ASCII value of the character.
3. Then, ask the user to form a sentence using both the character and its ASCII value.
4. Print the sentence, ensuring the use of the TAB escape sequence before it to format it neatly.

   Example:

   ```
   Suppose the character input is '@' and the user is informed of the corresponding ASCII value (64), then they might input the final sentence as: "In a world dominated by hashtags, the @ symbol, with an ASCII value of 64, quietly powers our email addresses."
   ```

Chapter 5: Understanding Truths in Life

Every life is filled with truths and myths. As in computers, it's essential to recognize truths. The idea here is to help your child understand the simplicity of truth, especially in the realm of numbers. Always be discerning and seek the truth!

1. Declare a variable of type bool named lifeTruth.
2. Prompt the user: "Enter a number. Let's explore its truthiness in the world of computers."
3. Take user input into an int variable, and assign it to lifeTruth.
4. Display the result as: "Truthiness of [int] in the world of computers: [lifeTruth * EARTH_GRAVITY]."

   Example:

   ```
   Enter a number. Let's explore its truthiness in the world of computers. 2 Truthiness of [int] in the world of computers: 9.81.
   ```

Chapter 6: Stringing Along Memories

In the tapestry of life, our memories are the threads that connect every moment, every emotion, and every lesson. Each memory is like a string, woven together to create the intricate designs of our experiences. As our child grows, we want to teach them the art of preserving memories, cherishing tales, and sharing their unique stories with the world.

1. Ask the user: "What's your child's first word going to be?"
2. Take input into a string.
3. Create a new string variable named `legacyMessage`. Use the `+` operator to concatenate the user's input with the `WISDOM` constant in a meaningful sentence.
4. Display `legacyMessage` as the final output: "Your child's legacy will be tied with [word] and the values of [WISDOM]."
5. Example output:

```
        What's your child's first word going to be? Bitcoin
        Your child's legacy will be tied with Bitcoin and the values of
Love, Kindness, and Integrity. And always eat your vegetables!
```

Chapter 7: The Value of Money

In the imaginative world of Academia, transactions and financial lessons are taught using Academia Dollars (ACD), a currency designed to impart lessons on money management and the importance of budgeting.

1. Ask the user to enter an amount in US Dollars (USD) they wish to convert to ACD. The amount entered should be a whole number stored in an integer type variable.
2. Use the given exchange rate to convert the USD amount to ACD. Assume the exchange rate is ACD = (USD/2) * 0.85.
3. Output the equivalent amount in ACD, formatted to two decimal places.
4. Example output:

```
        Please enter the amount in USD you wish to convert to ACD: 100
        Your 100 USD converts to 42.50 ACD in the world of Academia.
```

Tips and Notes:

1. **Main Function Only:** All your code should be enclosed within the `main` function (except for global constants). You may **not** create additional functions for this assignment.
2. **Use of Data Types and Operators**: Your program should make comprehensive use of the data types (short, int, float, double, char, string, bool, etc.) and operators (+, -, *, /, etc.) discussed in recent lectures. Pay particular attention to scenarios where one data type might be more appropriate than another to prevent data overflows.

3. **Static Casting:** Remember to use `static_cast` to prevent unintended results in arithmetic operations.
4. **Handling Strings with Whitespaces**: Remember to use the getline() function for inputs that might contain whitespaces. Note that `cin.ignore()` is required if the `getline()` function is used *after* a standard `cin` operation anywhere in your program. Its purpose is to clear out any lingering characters in the input buffer (e.g., `cin` tends to leave behind a newline character in the buffer after the user hits ENTER from the keyboard).

   ```
   #include <string> // required for using string and getline()
   …
   int studentNumber;
   string fullName;
   cout << "Please enter your student number: ";
   cin >> studentNumber; // input: 12345678
   cout << "\nPlease enter your full name: ";
   cin.ignore(); // clear the '\n' character in the input buffer
   getline(cin, fullName); // input: Mary Thompson
   cout << "Hello, " << fullName << "!" << endl;
   ```

   Note: If there have been no previous inputs using `cin`, then you MUST NOT use `cin.ignore()`before calling `getline()`. Otherwise, it might lead to unexpected input behavior.
5. **Formatting Output to Two Decimal Places:** In C++, this can be achieved using the `iomanip` library.
   1. Include the `iomanip` standard library at the beginning of your program.
   2. Use the `setprecision` and `fixed` manipulators to set the decimal precision of the output to ensure that exactly two digits appear after the decimal point.
      ```
      cout << std::fixed << std::setprecision(2) << yourVariable;
      ```
   3. This effect is persistent for the remaining C++ program operations until the formatting is changed again. If you want to revert back to the default formatting, you could do:
      ```
      cout.unsetf(ios::fixed);
      cout << std::setprecision(6);
      ```
6. **User Interactivity**: Your program should interact with the user, utilizing cin and cout to take inputs and display outputs respectively. Ensure your prompts are clear and your outputs are formatted neatly.
7. **No Error Handling Needed:** Do not worry about input validations or error handling for this assignment. Assume all user inputs will be valid.
8. **Review and Test:** Before submitting, review your code to ensure you've covered all requirements. Test your program with various inputs to ensure it behaves as expected.

Grading Criteria:

1. Clarity and Understandability (20%)
    - Code should be easy to follow.
    - Variables are appropriately named and clearly defined.
    - Logical flow is evident.
2. Completeness (20%)
    - All components of the assignment are addressed.
    - Edge cases or unique scenarios are considered and handled appropriately.
3. Optimization and Efficiency (20%)
    - The solution takes advantage of efficient methods and avoids unnecessary steps.
    - The most appropriate data types for variables are selected in the program.
4. Syntax and Structure (15%)
    - Code follows a consistent format and adheres to C++ standards.
    - Proper indentation is used to enhance code readability.
    - Use of appropriate C++ conventions and constructs.
5. Problem-Solving and Logic (15%)
    - The solution effectively addresses the problem.
    - Logic used in the code is sound and free of major errors.
6. Documentation and Comments (5%)
    - Important steps are commented for clarity.
    - Complex sections of code have accompanying explanations.
7. Creativity and Originality (5%)
    - Students showcase original thinking.
    - Unique and efficient solutions to problems are encouraged.